基于形式源级转换的可执行规格说明技术

张幸儿 朱晓军

(南京大学计算机科学系，南京 210008) (华北计算技术研究所，北京 100083)

AN APPROACH TO EXECUTABLE SPECIFICATIONS BASED ON FORMAL SOURCE_TO_SOURCE CONVERSION

Zhang Xinger

(Department of Computer Science, Nanjing University, Nanjing 210008)

Zhu Xiaojun

(North_China Institute of Computing Technology, Beijing 100083)

Abstract The paper proposes a new technique for executable specifications, which directly converts a specification into a program in a programming language by taking advantage of the idea of source_to_source conversion which is based on the rules defined formally and the canonical abstract syntax trees. The technique and the corresponding supporting system possess the following virtues: supporting rapid prototyping and getting the effects of rapid prototypes well.

摘要 本文提出一种新的可执行规格说明实现技术，它利用源级转换思想把规格说明直接转换成程序设计语言程序，且这种源级转换基于形式定义的规则与基于规范抽象语法树．该技术及相应的支持系统支持速成原型技术，具有支持系统开发工作量小、易保证正确性、开发代价低且开发周期短等特点，能较好地达到速成原型的效果．

§1. 引 言

速成原型技术对软件的设计与开发产生了重大影响，它是向经典的瀑布型软件生命周期模型的挑战．软件开发人员与用户通过速成原型可以在软件开发早期了解待开发软件的性行，检查它是否满足或符合用户的需求，或者使用户需求更明确．速成原型技术

本文 1990 年 10 月定稿．作者 张幸儿，南京大学副教授，主要从事程序设计语言与编译，程序设计方法学，软件工程工具环境方面的研究工作． 朱晓军，助工，1990 年硕士毕业于南京大学，目前主要从事计算机科学，软件工程方面的研究工作．

的重要特征是速成与廉价．开发一个软件系统时，常常关于系统的功能需求建立原型．速成原型的一个有效实现途径是可执行规格说明 [1][2][3].

一个形式规格说明用规格说明语言写出，规格说明语言具有形式化、表达明确与无二义性等特点．但关于功能需求的规格说明不一定是可执行的，这时需要把它翻译成一个实现．引起人们较大兴趣的是可执行规格说明．当一个规格说明能以某种方式 "执行" 时，它就提供了待开发软件的一个速成原型版本．可执行规格说明通常采用代数规格说明技术与有限状态模型技术．但迄今对于可执行规格说明均需要有简化系统或解释系统作为支持系统以 "执行" 它们．这些支持系统本身的开发需要一定的时间与代价，且 "执行" 的原型效果因支持系统而异.

本文提出一种新的可执行规格说明实现技术，它利用异种高级程序设计语言间源级转换思想，把规格说明直接转换到程序设计语言程序，且这种源级转换基于形式定义的语法规则与转换规则 [4]，并使用规范抽象语法树 (缩写为 CAST) 作为中间表示 [5]．这种技术的优点是开发支持系统的代价低廉且能获得较好的速成原型效果.

§ 2. 源级转换与可执行规格说明

2.1 源级转换

源级转换是指两个高级程序设计语言间的源程序级转换，也即把一个语言 A 的源程序 P_A 转换成另一种语言 B 的等价源程序 P_B．所谓等价，是指功能等价，也即 $[P_A]=[P_B]$，表示作 $P_A \equiv P_B$.

在实现源级转换时，关键是找出两种语言间的对应关系．当这对应关系被正确地确定并实现时，P_B 的性行完全反映了 P_A 的性行.

一个形式规格说明语言作为一种甚高级程序设计语言，具有通常高级程序设计语言的基本特征，有语法、语义与语用三方面．如果说，能找到一种规格说明语言 A 与某种高级程序设计语言 B 之间的对应关系，便可将 A 所写规格说明 S_A 转换成等价的 B 程序 P_B，即 $S_A \equiv P_B$，这样规格说明 S_A 无需被翻译成某个实现便可被 "执行"．对应关系完全可事先唯一地甚至形式地确定，转换的正确性是易于保证的．执行 P_B 时所呈现的性行将反映 S_A 中所描述的待开发软件的性行.

2.2 可执行规格说明语言的设计

规格说明用来描述一个软件系统 "做什么"，而不描述 "如何做"，规格说明语言应该是一个说明性语言，而且容易用来书写易读的规格说明.

一个规格说明语言，如同通常的程序设计语言，其设计的好坏直接影响到表达能力的强弱．一个规格说明语言不必要也不可能是能满足任何应用领域要求的万能语言．在一定的范围与程度上，一个规格说明语言是表达能力与实现代价等因素的折衷.

为了能应用源级转换技术于可执行规格说明语言，在设计规格说明语言时，应该事先确定一个合适的 "目标" 高级程序设计语言，用这种规格说明语言所写的规格说明是易于转换成该 "目标" 语言的源程序的.

为了能有利于应用自顶向下逐步精化设计技术和有利于软件可重用性，一个规格说明语言最好是模块化的.

基于上述考虑，设计了规格说明语言 NCSL，它以下列两点为其设计指导思想： 1)

模块化， 2) 基于逻辑且与逻辑程序设计语言相联系.

§ 3. NCSL 的设计及其支持系统

3.1 NCSL 的简况

NCSL 是在前述设计指导思想下设计的，它是一个基于逻辑的、模块化的且可执行的规格说明语言.

NCSL 具有操作语义，也即它不是通过表达函数之间关系的等式来描述系统的功能需求，而是用执行的效果来描述. 且看下列例子.

对于插入分类软件，可以有下列 NCSL 规格说明 (片断):

```
predicate insorted(list& L1, list& L2)
{ list L3;
   return (L1==nillist && L2==nillist
           ||
           insorted(tl(L1),L3) && insert(hd(L1),L3,L2)
           );
}
```

该例以一阶谓词形式来描述，其中的 insorted 与 insert 都是谓词，因此说是基于逻辑的. 事实上，用 NCSL 写的规格说明将被源级转换到 PROLOG 程序. 例中第一行括号对中指明的参数 L1 是初始数据，而 L2 是执行结果，其操作语义是明显的，即执行 insorted 的结果是:

当 L1 为空表列 (nillist) 时， L2 也是空表列;

当 L1 为非空表列时，先调用执行 insorted(tl(L1),L3), 使 L3 是对 L1 除第一个元素外的尾部进行插入分类的结果，然后调用执行 insert(hd(L1),L3,L2), 把 L1 的第一个元素插入已插入分类过的 L3 的适当位置上，从而在 L2 中得到对 L1 插入分类的结果.

为了便于利用这一插入分类软件规格说明，避免重复地书写，可以把它及其它有关成分处理作一个模块，这时完整的规格说明呈下列结构:

```
module insert_sorted;
      exports insorted;
      predicate insert(···)
      {···}
      predicate insorted(···)
      {···}
      void sort(list& L1,list& L2)
      { pre (true);
        post (insorted(L1,L2));
      }
end
```

3.2 支持系统的设计与实现

1) 源级转换实现途径

一个 NCSL 规格说明，首先基于形式定义的语法规则被转换成等价的 CAST, 然后基于形式定义的转换规则，从所生成的 CAST 生成相应的 PROLOG 程序. 规范抽象语

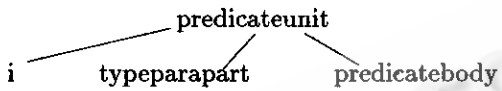
法树是关于按一定规则规范地生成的抽象语法 (规范抽象语法) 而生成的. 例如，对于用 BNF 表示的语法规则

predicateunit ::= predicate i (typeparapart) { predicatebody }

相应的规范抽象语法是：

predicateunit i typeparapart predicatebody

其中弃去了与表示法有关的信息 (关键词 predicate 与各类括号). 相应的规范抽象语法树呈下形：



形式定义的转换规则呈下形：

⇒ @ is predicate i (typeparapart) predicatebody

符号 "⇒" 标志其后是转换规则，即把相应 BNF 规则的右部所对应规格说明部分 (相应 CAST) 按此转换规则进行转换. 首先处理@.is predicate, 它是一个系统子程序名，相应系统子程序的功能是为处理相应 predicate 作准备. 然后依次处理其后各符号：输出 i 所相应的实际标识符 (谓词名)、输出 "(" 、相应于 typeparapart 的部分及 ")" , 最后输出相应于 predicatebody 的部分. 关于 typeparapart 与 predicatebody 的处理类似于 predicateunit 的处理，即按相应的转换规则输出相应的 PROLOG 结构.

2) NCSL 支持系统的总体结构

NCSL 支持系统基于形式定义的规则与规范抽象语法树，它由三部分组成：规则处理器、CAST 生成器与树到 PROLOG 程序转换器. 总体结构示意图如图 1 所示.

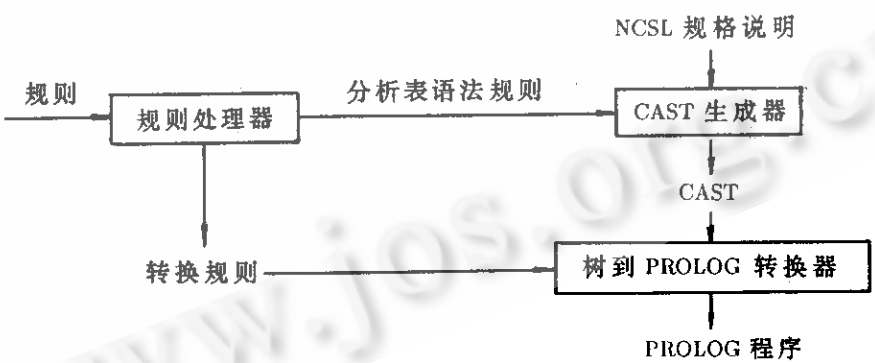


图 1

规则处理器：其功能是输入关于规格说明语言 NCSL 的语法规则及从 NCSL 到 PROLOG 的转换规则. 语法规则采用 BNF 表示法表示，而转换规则类似于 BNF 表示法的规则右部，只是可能包含有以@打头的系统子程序名. 规则处理器生成相应于语法规则的 LR 分析表，同时得到规则的内部表示形式. 规则处理器仅工作一次，一旦生成了 LR 分析表及规则的内部表示，规则处理器即可弃去.

CAST 生成器：它以 NCSL 规格说明作为输入，基于语法规则 (也即 LR 分析表) 把规格说明转换成等价的 CAST, 作为树到 PROLOG 程序转换器的输入.

树到 PROLOG 程序转换器：其功能是根据转换规则从 CAST 生成相应的 PROLOG 程序，工作原理如下.

从 CAST 的根结点出发，找出相应于结点名的转换规则，顺次地处理该规则中的各个符号. 当是运算分量终结符号，输出相应的标识符或常量. 当是 PROLOG 终结符号时立即输出它. 当是 NCSL 非终结符号时，把当前转换规则及其当前处理位置下推入栈，并把相应于非终结符号的子结点上指明的转换规则作为当前处理的. 在此当前转换规则中的符号类似地依次处理.

系统子程序用于处理两种语言间既不可直接对应，也不可间接对应的成分. 当处理到的转换规则符号是系统子程序名时便调用相应的子程序，结束时返回到处理转换规则中相邻的下一个符号.

当一个转换规则的一切符号依次被处理过也即达到其右端时，栈顶上的转换规则成为当前处理的，从刚处理过的符号的相邻下一个符号开始，类似地进行处理. 如此继续，直到栈中再无规则要被处理.

树到 PROLOG 程序转换器的控制流程示意图如图 2 所示.

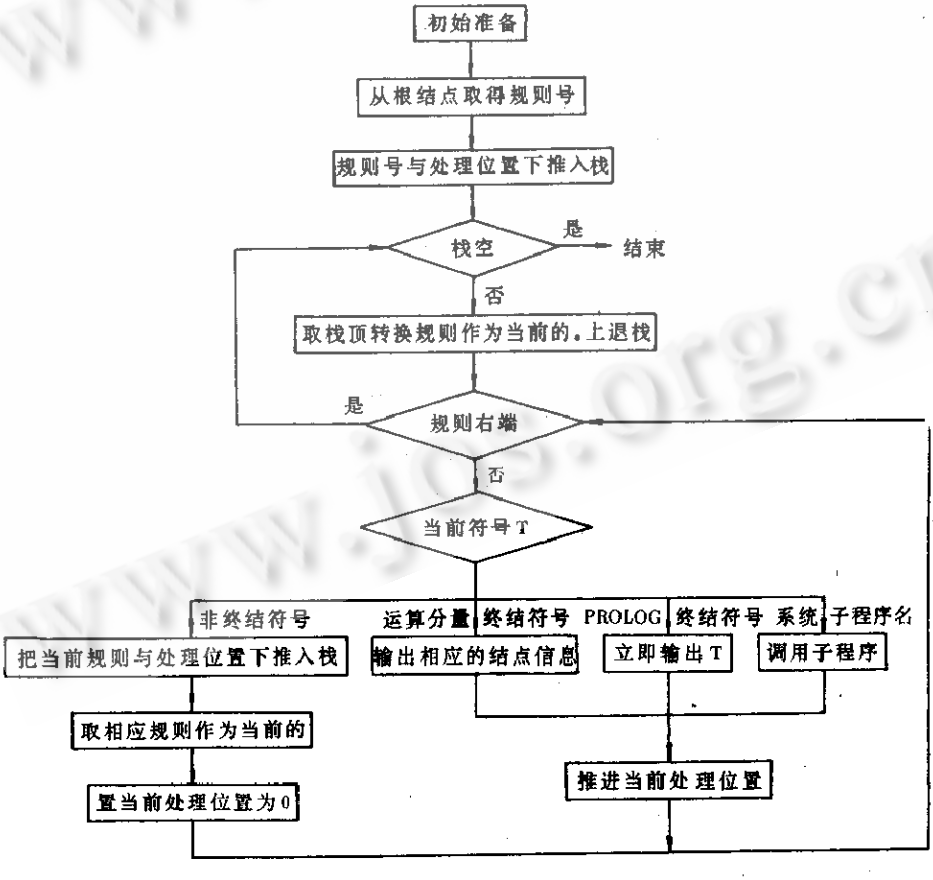


图 2

一个 PROLOG 程序应是有格式的，以便于阅读．一般地可以由漂亮格式打印器来完成漂亮格式处理，当前实现中由适当的系统子程序来完成．

§ 4. 从 NCSL 到 PROLOG 转换中的问题

4.1 规格说明与程序的结构

规格说明中以谓词为单位，这些谓词遵循先定义后使用的原则，因此，在谓词 insorted 的定义中引用了谓词 insert，后者的定义必须出现在 insorted 的定义之前．

然而对于 PROLOG 程序来说，关于 insorted 的事实与规则必须出现在关于谓词 insert 的事实与规则之前．

例如，关于插入分类例子，下列谓词定义：

```
predicate insert (generic& A, list& L1, list& L2)
{ list L3;
  return ( L1==nillist && L2==cons(A,nillist)
           ||
           A<=hd(L1) && L2==cons(A,L1)
           ||
           insert(A,tl(L1),L3) && L2==cons(hd(L1),L3)
           );
}
```

必须出现在 § 3 的 3.1 中的谓词 insorted 的定义前面．然而相应的 PROLOG 程序将呈下形．

```
insorted([ ],[ ]).
insorted([H1|T1],L2) :— insorted(T1,L3), insert(H1,L3,L2).
insert(A,[ ],[A]).
insert(A,[H1|T1],[A|[H1|T1]]) :— A=<H1.
insert(A,[H1|T1],[H1|L3]) :— insert(A,T1,L3).
```

按照基于形式定义的源级转换思想，结构的转换借助于下列规则实现：

```
predicatepart::=predicatepart predicateunit
  ⇒ @ mostright predicateunit @ leftnext predicatepart
```

相应的 (改进了的)CAST 如图 3 所示．其中@ mostright 的功能是取到最右子结点，在处理 predicateunit 符号时使得处理的是最右子结点，在插入分类例中对应于谓词 insert．@ leftnext 的功能是取到刚处理的结点的左边一个结点，在插入分类例中对应于谓词 insorted．

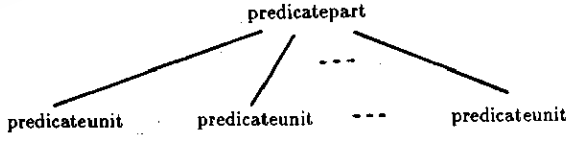


图 3

4.2 变量赋值与同一化操作

一个规格说明描述做什么，不是描述如何做，因此不需也不应该考虑变量存贮分配等具体实现问题.

冯. 诺伊曼型程序设计风格的特征却是基于存贮字的概念，必须考虑如何指明变量的属性与如何分配相应大小的内存区域，从而实现对变量的算术运算、赋值与传输等.

PROLOG 语言提供了表列作为最常用的重要数据结构，在 PROLOG 程序中对对象的属性无需显式说明，且通过同一化操作对变量进行赋值. 因此，在 PROLOG 支持系统的支持下，把一个 NCSL 规格说明转换成 PROLOG 程序时对对象 (数据) 的处理显得相当简单和易于实现.

4.3 速成原型效果

一个可执行规格说明描述了一个待开发软件的性行. 此性行通常通过执行关于此规格说明的使用程序而获得. 不同的使用程序一般可反映不同方面的性行. 这意味着需要有一定数量的使用程序，在可执行规格说明语言支持环境下运行.

PROLOG 语言具有询问设施，用户只需键入询问语句，支持系统便可给出相应的回答信息. 例如，假定已把前述插入分类软件的 NCSL 规格说明源级转换成 PROLOG 程序. 现在键入下列询问语句： ? —insorted([ ],L2).

支持系统将给出下列回答： L2=[ ]

如果键入下列询问语句： ? —insorted([3,6,1],L).

支持系统将给出下列回答： L=[1,3,6]

显然利用 PROLOG 的询问设施可以快速地获取待开发软件的性行消息.

结语： 本文提出了一种利用源级转换思想把规格说明语言直接转换成程序设计语言的技术，它基于形式定义的语法规则与转换规则，并基于规范抽象语法树. 这为可执行规格说明提供了一种新的实现途径. 这一实现技术及相应的 NCSL 支持系统有如下几个特点. **1.** 规则形式定义且处理规范化，开发 NCSL 支持系统所需的工作量小，且易于保证正确性，因此支持系统开发成本低且开发周期短. **2.** 把 NCSL 规格说明转换成 PROLOG 程序，可利用 PROLOG 语言的特性，降低支持系统的复杂性，进一步减少系统开发的工作量. **3.** 利用 PROLOG 语言的询问设施，能方便快速地获取待开发软件的性行消息. 概括起来，这一技术与 NCSL 支持系统支持速成原型技术，达到较好的速成原型效果. NCSL 支持系统的开发过程体现了上述诸特点. 但这还是实验性的，是初步探索.

参考文献

[1] J.A.Goguen and J.Meseguer, Rapid Prototyping in the OBJ Executable Specification Language, Software Engineering Notes, Dec. 1982.

[2] S. Lee, On Executable Models for Rule_Based Prototyping, in Proceedings of 8th International Conference on Software Engineering, 1985.

[3] R.B.Terwilliger and P.A.Kirslis, PK/C+ + : An Object_Oriented, Logic_Based, Executable Specification Language, in Proceedings of the 22th International Conference on Systems Sciences, 1989.

[4] Zhang Xinger and Zhu Xiaojun et al., Source_to_Source Conversion Based on Formal Definition, Journal of Computer Science and Technology, 6:2(1991).

[5] 张幸儿，规范抽象语法与抽象语法树的直接生成，计算机学报， 13:12(1990).